# Project1-CSE3027-Spring 2024

2022098068

Kim YoungTaek

## 1. Server Design

The server design comprises the following steps:

1. **Socket Creation**: Create sockets for IPv4 and TCP environments.
2. **Address Binding**: Assign IP addresses and port numbers to sockets for binding.
3. **Connection Reception**: Wait for incoming connection requests from clients.
4. **Client Connection Acceptance**: Accept client connection requests and create client sockets.
5. **Data Transmission**: Receive data from clients, process it, and return the content of requested files.
6. **Loop Execution**: The server continuously loops from the acceptance phase to closing the client socket, ensuring continuous reception of requests from clients.

Before opening the server, it receives the following arguments:

- If no port number is provided, an error occurs, and the program terminates.
- Upon successful reception of the port number, sockets for IPv4 and TCP environments are created. Then, IP addresses and port numbers are assigned to the sockets. Additionally, the socket's option is set using the setsockopt() function to ensure immediate termination of the server upon program termination.

Once the address binding is successfully completed, the server waits for client connections. Upon successful establishment of a connection, the server reads the client's request and processes it accordingly, including opening and returning the content of requested files. After completing these steps, the server closes the file and the client socket. Then, it accepts the client socket again and repeats the above steps until the user forcibly terminates the program.

## 2. Difficulties

The first challenge was error handling. Effectively managing various errors that can occur during socket operations such as creation, binding, listening, and accepting connections was necessary. These operations can often fail due to issues like the port being already in use, insufficient permissions, or network errors. To address this, I immediately checked the error codes after each socket operation and printed appropriate error messages to safely terminate the program if needed.

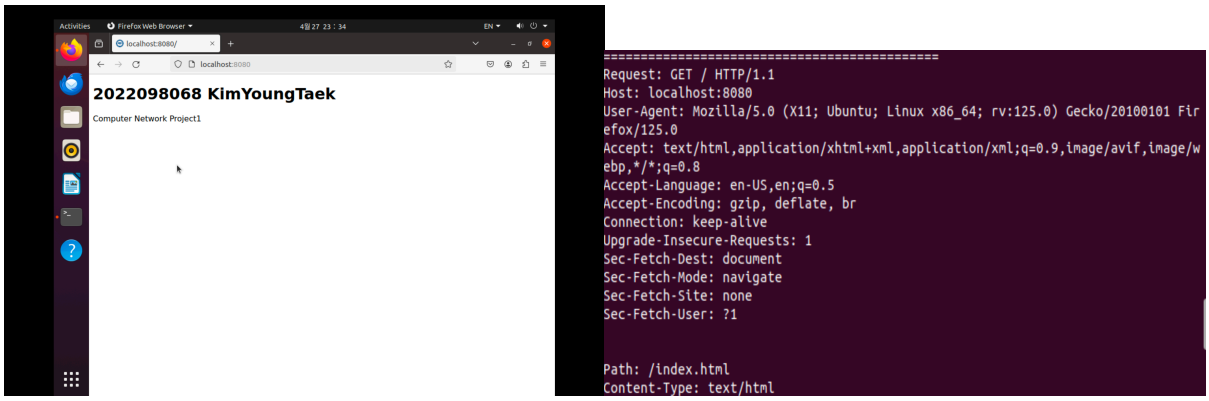
The second challenge involved parsing HTTP requests. It was necessary to correctly parse the HTTP requests received from clients to extract the needed information. Handling syntax errors or unexpected inputs during this process was a significant challenge. I used simple string functions to extract the file path from GET requests, and for invalid requests, I returned a 400 Bad Request response to resolve this issue.

## 3. Sample Output

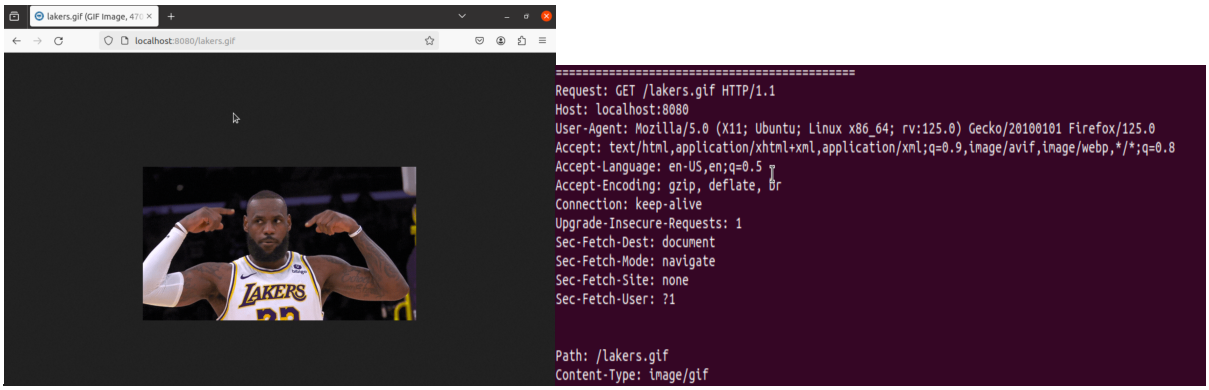
```
taek@taek:~/Desktop/ComputerNetwork$ ./server 8080
```

After opening a server on port 8080 using the above commands, here are the respective output values.

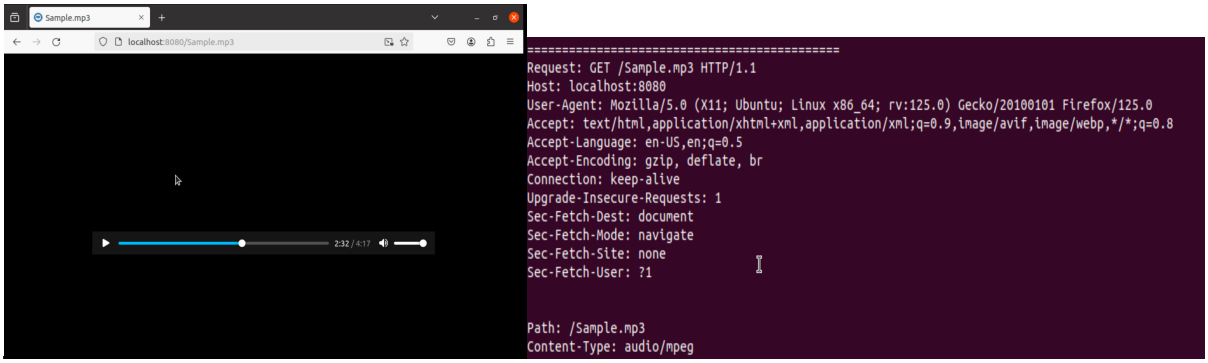
1) localhost:8080(index.html)



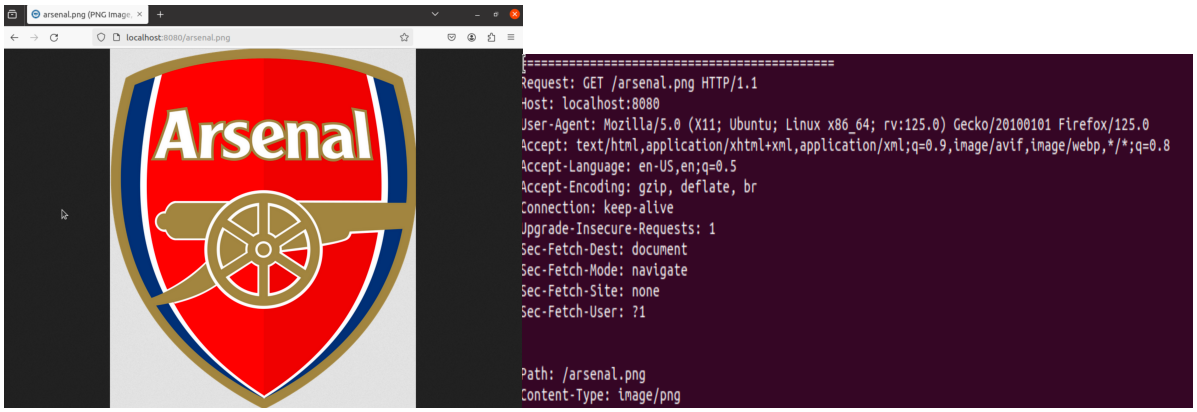

2) localhost:8080/lakers.gif




3) localhost:8080/Sample.mp3




4) localhost:8080/arsenal.png




5) localhost:8080/proj1.pdf